On Using MOSEK to Solve

Large-Scale Linear and Conic Optimization Problems

Erling D. Andersen

MOSEK ApS

INFORMS annual meeting
Minneapolis, October 6-9, 2013

Some facts

About MOSEK:

- Software package for solving optimization problems.
- Version 1 released 1999.
- Version 7 released 2013.

Problem types:

- Linear + integer variables.
- Conic quadratic + integer variables.
- Semidefinite optimization.
- Convex quadratic + integer variables.
- General convex.

Some facts

About MOSEK:

- Software package for solving optimization problems.
- Version 1 released 1999.
- Version 7 released 2013.

Problem types:

- Linear + integer variables.
- Conic quadratic + integer variables.
- Semidefinite optimization.
- Convex quadratic + integer variables.
- General convex.

The system

Several interfaces:

- Fusion API, Optimizer API and toolbox.
- Supports different languages and tools.

One optimization engine:

- Written C.
- Tuned for the large-scale sparse case.
- Exploit hardware features such as AVX instructions.

The system

Several interfaces:

- Fusion API, Optimizer API and toolbox.
- Supports different languages and tools.

One optimization engine:

- Written C.
- Tuned for the large-scale sparse case.
- Exploit hardware features such as AVX instructions.

Optimizers for continuous problems

Optimizer	Problem type: Network	Linear	Conic	Convex
Network simplex	+			
Primal simplex	+	+		
Dual simplex	+	+		
Interior-point	+	+	+	+

Simplex optimizers

- Large-scale sparse.
- Many options for pricing etc.

Interior-point

- Large-scale sparse with tuned linear algebra.
- Parallelized.
- Reliable infeasibility detection and reporting.

Optimizers for continuous problems

Optimizer	Problem type: Network	Linear	Conic	Convex
Network simplex	+			
Primal simplex	+	+		
Dual simplex	+	+		
Interior-point	+	+	+	+

Simplex optimizers

- Large-scale sparse.
- Many options for pricing etc.

Interior-point

- Large-scale sparse with tuned linear algebra.
- Parallelized.
- Reliable infeasibility detection and reporting.

Optimizers for continuous problems

	Problem type			
Optimizer	Network	Linear	Conic	Convex
Network simplex	+			
Primal simplex	+	+		
Dual simplex	+	+		
Interior-point	+	+	+	+

Simplex optimizers

- Large-scale sparse.
- Many options for pricing etc.

Interior-point

- Large-scale sparse with tuned linear algebra.
- Parallelized.
- Reliable infeasibility detection and reporting.

Optimizers for mixed-integer problems

Mixed integer conic

- Solves mixed-integer linear and conic quadratic problems.
- Parallelized.
- Run-to-run deterministic.
- Tuned for conic quadratic problems.
- No additional charge.

Mixed integer optimizer

- Solves mixed-integer linear and conic quadratic.
- Tuned for linear problems.

Optimizers for mixed-integer problems

Mixed integer conic

- Solves mixed-integer linear and conic quadratic problems.
- Parallelized.
- Run-to-run deterministic.
- Tuned for conic quadratic problems.
- No additional charge.

Mixed integer optimizer

- Solves mixed-integer linear and conic quadratic.
- Tuned for linear problems.

Supported platforms and tools

Supported platforms operating systems

Windows,MAC OSX, Linux

MOSEK interfaces

AMPL, C/C++, Java, Python, Matlab, Microsoft .NET, R

Third party products

AIMMS, GAMS, Frontline Solver, CVX, Woodstock

Other interfaces

COIN OSI, Raven toolbox, Yalmip ...

Supported platforms and tools

Supported platforms operating systems

Windows,MAC OSX, Linux

MOSEK interfaces

AMPL, C/C++, Java, Python, Matlab, Microsoft .NET, R

Third party products

AIMMS, GAMS, Frontline Solver, CVX, Woodstock

Other interfaces

COIN OSI, Raven toolbox, Yalmip ...

Supported platforms and tools

Supported platforms operating systems

Windows,MAC OSX, Linux

MOSEK interfaces

AMPL, C/C++, Java, Python, Matlab, Microsoft .NET, R

Third party products

AIMMS, GAMS, Frontline Solver, CVX, Woodstock

Other interfaces

COIN OSI, Raven toolbox, Yalmip ...

Supported platforms and tools

Supported platforms operating systems

Windows,MAC OSX, Linux

MOSEK interfaces

AMPL, C/C++, Java, Python, Matlab, Microsoft .NET, R

Third party products

AIMMS, GAMS, Frontline Solver, CVX, Woodstock

Other interfaces

COIN OSI, Raven toolbox, Yalmip ...

The MOSEK solvers

Facts

Availability

Using MOSEK

Installation

An example: Portfolio optimization

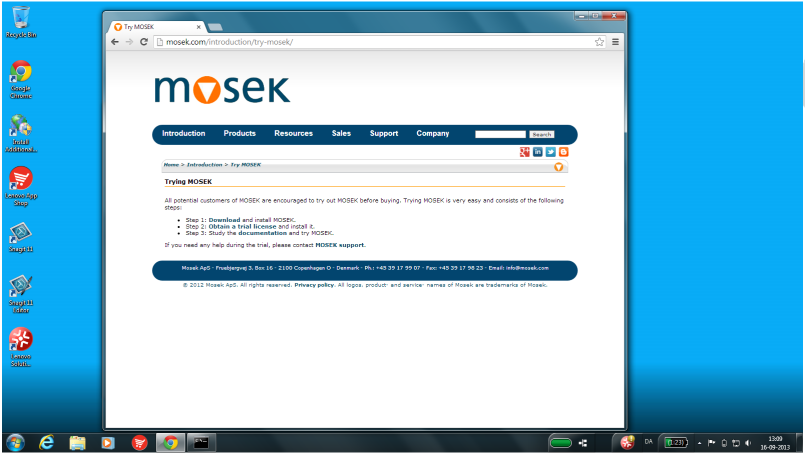
Portfolio optimization in Python Fusion

Portfolio optimization in MATLAB

Portfolio optimization: The optimizer API

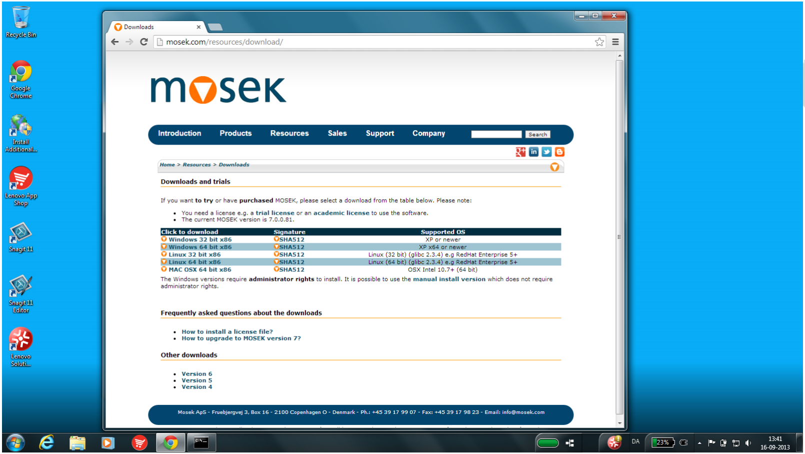
Concluding remarks

Try MOSEK

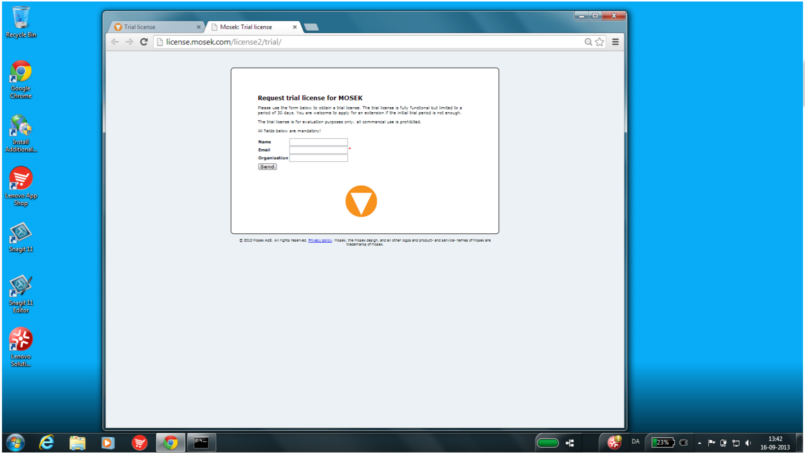


Try MOSEK
mosek.com/introduction/try-mosek/
Introduction
Products
Resources
Sales
Support
Company
Search
Home > Introduction > Try MOSEK
Trying MOSEK
All potential customers of MOSEK are encouraged to try out MOSEK before buying. Trying MOSEK is very easy and consists of the following steps:
Step 1: Download and install MOSEK.
Step 2: Obtain a trial license and install it.
Step 3: Study the documentation and try MOSEK.
If you need any help during the trial, please contact MOSEK support.
Mosek ApS - Fruebjergvej 3, Box 16 - 2100 Copenhagen O - Denmark - Ph.: +45 39 17 99 07 - Fax: +45 39 17 98 23 - Email: info@mosek.com
© 2012 Mosek ApS. All rights reserved. Privacy policy. All logos, product- and service- names of Mosek are trademarks of Mosek.
Recycle Bin
Google Chrome
Install Additional...
Lenovo App Shop
Snagit 11
Snagit 11 Editor
Lenovo Soluti...
13:09
16-09-2013

Download MOSEK



Obtain atrial license



Portfolio optimization

Markowitz portfolio optimization

- Find the optimal portfolio of assets.
- A one period model.
- Invented by H. Markowitz.
- Used extensively by hedge funds and investment companies.

The model

Markowitz model:

$$
\begin{array}{ll}
\text{maximize} & r^T x - \alpha s \\
\text{subject to} & \sum_i x_i = 1 \\
 & (s, Gx) \in \mathcal{Q}^n \\
 & x \geq 0,
\end{array}
$$

with r: average return, $G^T G$: correlation, α: risk-aversion.

- $(s, Gx) \in \mathcal{Q}^n \quad \Leftrightarrow \quad s \geq ||Gx||.$
- s is the std. dev. of the return i.e. risk.
- Optimize a weighted combination of return and risk.
- In practice solved for many values of α.

The model

Markowitz model:

$$
\begin{array}{ll}
\text{maximize} & r^T x - \alpha s \\
\text{subject to} & \sum_i x_i = 1 \\
 & (s, Gx) \in \mathcal{Q}^n \\
 & x \geq 0,
\end{array}
$$

with r: average return, $G^T G$: correlation, α: risk-aversion.

- $(s, Gx) \in \mathcal{Q}^n \quad \Leftrightarrow \quad s \geq ||Gx||$.
- s is the std. dev. of the return i.e. risk.
- Optimize a weighted combination of return and risk.
- In practice solved for many values of α.

The model

Markowitz model:

$$
\begin{array}{ll}
\text{maximize} & r^T x - \alpha s \\
\text{subject to} & \sum_i x_i = 1 \\
& (s, Gx) \in \mathcal{Q}^n \\
& x \geq 0,
\end{array}
$$

with r: average return, $G^T G$: correlation, α: risk-aversion.

- $(s, Gx) \in \mathcal{Q}^n \quad \Leftrightarrow \quad s \geq ||Gx||.$
- s is the std. dev. of the return i.e. risk.
- Optimize a weighted combination of return and risk.
- In practice solved for many values of α.

The model

Markowitz model:

$$
\begin{array}{ll}
\text{maximize} & r^T x - \alpha s \\
\text{subject to} & \sum_i x_i = 1 \\
& (s, Gx) \in \mathcal{Q}^n \\
& x \geq 0,
\end{array}
$$

with r: average return, $G^T G$: correlation, α: risk-aversion.

- $(s, Gx) \in \mathcal{Q}^n \quad \Leftrightarrow \quad s \geq ||Gx||$.
- s is the std. dev. of the return i.e. risk.
- Optimize a weighted combination of return and risk.
- In practice solved for many values of α.

The model

Markowitz model:

$$
\begin{array}{ll}
\text{maximize} & r^T x - \alpha s \\
\text{subject to} & \sum_i x_i = 1 \\
& (s, Gx) \in \mathcal{Q}^n \\
& x \geq 0,
\end{array}
$$

with r: average return, $G^T G$: correlation, α: risk-aversion.

- $(s, Gx) \in \mathcal{Q}^n \quad \Leftrightarrow \quad s \geq ||Gx||$.
- s is the std. dev. of the return i.e. risk.
- Optimize a weighted combination of return and risk.
- In practice solved for many values of α.

A MOSEK Fusion implementation

What is Fusion?

- An object orientated interface for building linear and conic optimization models.
- Works directly with variables and constraints.
- Easy to build and modify a model.
- Available for Java, MATLAB, .NET and Python.
- Models are similar in all languages.

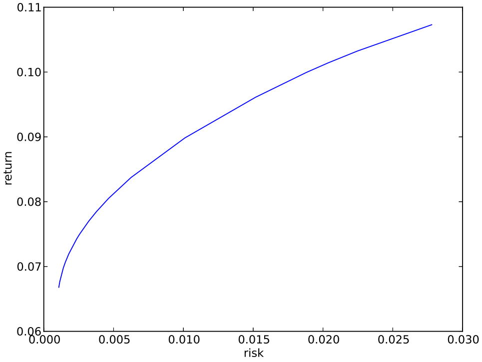
Portfolio example in Python Fusion:

```python
from mosek.fusion import *

def dot(x,y):
    r = 0.0
    for j in range(len(x)):
        r = r+x[j]*y[j]
    return r

def EfficientFrontier(r,G,alphas):
    n = len(r)

    M = Model('Efficient frontier')
    x = M.variable('x', n, Domain.greaterThan(0.0)) # Portfolio variables
    s = M.variable('s', 1, Domain.unbounded())      # Risk variable
    M.constraint('budget', Expr.sum(x), Domain.equalsTo(1.0))                     # sum(x) = 1
    M.constraint('risk',   Expr.vstack(s, Expr.mul(G,x)), Domain.inQCone())    # norm(Gx) <= s

    frontier = []
    for a in alphas:
       #  objective:  r'*x - a*s
       M.objective('obj', ObjectiveSense.Maximize, Expr.sub(Expr.dot(r,x), Expr.mul(a,s)))
       M.solve()
       frontier.append( (a, dot(r,x.level()), s.level()[0]) )

    return frontier

if __name__ == '__main__':

    r =   [ 0.1073,  0.0737,  0.0627 ]      # Vector of average returns
    G = [ [ 0.1667,  0.0232,  0.0013 ],     # Cholesky Factor of Sigma.
          [ 0.0000,  0.1033, -0.0022 ],
          [ 0.0000,  0.0000,  0.0338 ] ]

    alphas   = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1.0, 1.5, 2.0, 3.0, 10.0]
    frontier = EfficientFrontier(r,DenseMatrix(G),alphas)

    print('\nEfficient frontier')
    print('%-12s  %-12s  %-12s' % ('alpha','return','risk'))
    for i in frontier:
        print('%-12.4f  %-12.4e  %-12.4e' % (i[0],i[1],i[2]))
```

Portfolio optimization

Efficient Frontier

The *efficient frontier* shows the optimal trade-off.



Fusion summary

- Java, MATLAB and .NET Fusion looks almost identical.
- Model is close to the paper version.
- Excellent for rapid linear and conic model building.

MATLAB toolbox implementation

MATLAB

- MATLAB is a high-level language and interactive environment for numerical work.
- Popular among engineers, in finance, everywhere

MOSEK optimization toolbox for MATLAB includes

- A matrix orientated interface.
- Lower level than Fusion.
- linprog, quadprog, etc clones.

MATLAB toolbox implementation

MATLAB

- MATLAB is a high-level language and interactive environment for numerical work.
- Popular among engineers, in finance, everywhere

MOSEK optimization toolbox for MATLAB includes

- A matrix orientated interface.
- Lower level than Fusion.
- linprog, quadprog, etc clones.

Serializing the model

First reformulation:

$$
\begin{array}{ll}
\text{maximize} & r^T x - \alpha s \\
\text{subject to} & e^T x = 1 \\
 & Gx - t = 0 \\
 & (s, t) \in \mathcal{Q}^n \\
 & x \geq 0,
\end{array}
$$

where

$$e = [1, \ldots, 1]^T.$$

A new variable:

$$\bar{x} = \begin{bmatrix} x \\ t \\ s \end{bmatrix} = [x; t; s]$$

Serializing the model

First reformulation:

$$
\begin{array}{ll}
\text{maximize} & r^T x - \alpha s \\
\text{subject to} & e^T x = 1 \\
 & Gx - t = 0 \\
 & (s, t) \in \mathcal{Q}^n \\
 & x \geq 0,
\end{array}
$$

where

$$e = [1, \ldots, 1]^T.$$

A new variable:

$$\bar{x} = \begin{bmatrix} x \\ t \\ s \end{bmatrix} = [x; t; s]$$

Serializing the model

First reformulation:

$$\begin{array}{ll} \text{maximize} & r^T x - \alpha s \\ \text{subject to} & e^T x = 1 \\ & Gx - t = 0 \\ & (s, t) \in \mathcal{Q}^n \\ & x \geq 0, \end{array}$$

where

$$e = [1, \ldots, 1]^T.$$

A new variable:

$$\bar{x} = \begin{bmatrix} x \\ t \\ s \end{bmatrix} = [x; t; s]$$

cont.

Next reformulation

$$
\begin{array}{ll}
\text{maximize} & \begin{bmatrix} r^T & 0_{1\times n} & -\alpha \end{bmatrix} \bar{x} \\
\text{subject to} & \begin{bmatrix} e_{1\times n} & 0_{1\times n} & 0 \end{bmatrix} \bar{x} = 1 \\
& \begin{bmatrix} G & -I_{n\times n} & 0_{n\times 1} \end{bmatrix} \bar{x} = 0 \\
& \bar{x}_{2n+1} \geq ||\bar{x}_{(n+1):(2n)}|| \\
& \bar{x}_{1:n} \geq 0,
\end{array}
$$

MOSEK model

$$
\begin{array}{ll}
\text{maximize} & c^T x \\
\text{subject to} & l^c \leq Ax \leq u^c \\
& x \in K \\
& l^x \leq x \leq u^x
\end{array}
$$

cont.

Next reformulation

$$
\begin{array}{ll}
\text{maximize} & \begin{bmatrix} r^T & 0_{1\times n} & -\alpha \end{bmatrix} \bar{x} \\
\text{subject to} & \begin{bmatrix} e_{1\times n} & 0_{1\times n} & 0 \end{bmatrix} \bar{x} = 1 \\
& \begin{bmatrix} G & -I_{n\times n} & 0_{n\times 1} \end{bmatrix} \bar{x} = 0 \\
& \bar{x}_{2n+1} \geq \left|\left|\bar{x}_{(n+1):(2n)}\right|\right| \\
& \bar{x}_{1:n} \geq 0,
\end{array}
$$

MOSEK model

$$
\begin{array}{ll}
\text{maximize} & c^T x \\
\text{subject to} & l^c \leq Ax \leq u^c \\
& x \in K \\
& l^x \leq x \leq u^x
\end{array}
$$

cont.

Next reformulation

$$
\begin{array}{ll}
\text{maximize} & \begin{bmatrix} r^T & 0_{1\times n} & -\alpha \end{bmatrix} \bar{x} \\
\text{subject to} & \begin{bmatrix} e_{1\times n} & 0_{1\times n} & 0 \end{bmatrix} \bar{x} = 1 \\
 & \begin{bmatrix} G & -I_{n\times n} & 0_{n\times 1} \end{bmatrix} \bar{x} = 0 \\
 & \bar{x}_{2n+1} \geq ||\bar{x}_{(n+1):(2n)}|| \\
 & \bar{x}_{1:n} \geq 0,
\end{array}
$$

MOSEK model

$$
\begin{array}{ll}
\text{maximize} & c^T x \\
\text{subject to} & l^c \leq Ax \leq u^c \\
 & x \in K \\
 & l^x \leq x \leq u^x
\end{array}
$$

Portfolio example in MATLAB:

```
[ret, res] = mosekopt('symbcon echo(0)');

r          = [ 0.1073,  0.0737,  0.0627 ]';
G          = [ [ 0.1667,  0.0232,  0.0013 ];...
               [ 0.0000,  0.1033, -0.0022 ];...
               [ 0.0000,  0.0000,  0.0338 ] ];
alphas     = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1.0, 1.5, 2.0, 3.0, 10.0];
n          = length(r);

clear prob;

% The the problem.
prob.a            = [[ones(1,n),zeros(1,n),0];...
                     [G,-speye(n),zeros(n,1)]];
prob.blc          = [1;zeros(n,1)];
prob.buc          = [1;zeros(n,1)];
prob.blx          = [zeros(n,1);-inf*ones(n+1,1)];

prob.cones.type   = [res.symbcon.MSK_CT_QUAD];
prob.cones.sub    = [(2*n+1),(n+1):(2*n)];
prob.cones.subptr = [1];

% Compute the efficient frontier.
for i=1:length(alphas)
    alpha               = alphas(i);
    prob.c              = [r;zeros(n,1);-alpha];
    [ret,res]           = mosekopt('maximize echo(0)',prob);
    x                   = res.sol.itr.xx;
    fprintf('%.2e %.4e %.4e\n',alpha,r'*x(1:n),x(2*n+1));
end
```

MATLAB run

```
>> portfolio
alpha        ret          risk
0.00e+00     1.0730e-01   7.2173e-01
1.00e-01     1.0730e-01   1.6670e-01
2.00e-01     1.0730e-01   1.6670e-01
3.00e-01     8.0540e-02   6.8220e-02
4.00e-01     7.1951e-02   4.2329e-02
5.00e-01     6.9756e-02   3.7355e-02
7.50e-01     6.7660e-02   3.3827e-02
1.00e+00     6.6790e-02   3.2811e-02
1.50e+00     6.5984e-02   3.2139e-02
2.00e+00     6.5601e-02   3.1916e-02
3.00e+00     6.5221e-02   3.1758e-02
1.00e+01     6.4698e-02   3.1645e-02
```

MATLAB summary

- Matrix orientated input.
- Models must be serialized.
- Can be used to replace linprog and friends.
- Similar interface is available for R.

Using the optimizer API

API model:

$$
\begin{array}{ll}
\text{maximize} & c^T x \\
\text{subject to} & l^c \leq Ax \leq u^c \\
 & x \in K \\
 & l^x \leq x \leq u^x
\end{array}
$$

- Serialized view. One variable only.
- Use function calls to input data.

MATLAB example thinking reuse

Problem

$$
\begin{array}{ll}
\text{maximize} & \left[\begin{array}{ccc} r^T & 0_{1\times n} & -\alpha \end{array}\right] \bar{x} \\
\text{subject to} & \left[\begin{array}{ccc} e_{1\times n} & 0_{1\times n} & 0 \end{array}\right] \bar{x} = 1 \\
& \left[\begin{array}{ccc} G & -I_{n\times n} & 0_{n\times 1} \end{array}\right] \bar{x} = 0 \\
& \bar{x}_{2n+1} \geq ||\bar{x}_{(n+1):(2n)}|| \\
& \bar{x}_{1:n} \geq 0,
\end{array}
$$

```c
#include <math.h>
#include <stdio.h>

#include "mosek.h"

#define MOSEKCALL(_r,_call)  ( (_r)==MSK_RES_OK ? ( (_r) = (_call) ) : ( (_r) = (_r) ) );


static void MSKAPI printstr(void *handle,
                            MSKCONST char str[])
{
  printf("%s",str);
} /* printstr */


int main(int argc, const char  argv[])
{
  const MSKint32t n=3,numalpha=12;
  const double    r[]={0.1073,  0.0737,  0.0627},
                  G[][3]={{0.1667,  0.0232,  0.0013},
                          {0.0000,  0.1033, -0.0022},
                          {0.0000,  0.0000,  0.0338}},
                  alphas[12]={0.0, 0.1, 0.2, 0.3, 0.4, 0.5,
                              0.75, 1.0, 1.5, 2.0, 3.0, 10.0};
  MSKenv_t        env;
  MSKint32t       k,i,j,*sub;
  MSKrescodee     res=MSK_RES_OK;
  MSKtask_t       task;

  sub = calloc(n,sizeof(MSKint32t));

  res = sub==NULL ? MSK_RES_ERR_SPACE : MSK_RES_OK;
```

```c
MOSEKCALL(res,MSK_makeenv(&env,NULL));
MOSEKCALL(res,MSK_maketask(env,0,0,&task));
MOSEKCALL(res,MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr));

/* Constraints. */
MOSEKCALL(res,MSK_appendcons(task,1+n));
MOSEKCALL(res,MSK_putconbound(task,0,MSK_BK_FX,1.0,1.0));
for(i=0; i<n; ++i)
  MOSEKCALL(res,MSK_putconbound(task,1+i,MSK_BK_FX,0.0,0.0));

/* Variables. */
MOSEKCALL(res,MSK_appendvars(task,1+2*n));
/* x variables. */
for(j=0; j<n; ++j)
{
  MOSEKCALL(res,MSK_putcj(task,j,r[j]));
  MOSEKCALL(res,MSK_putaij(task,0,j,1.0));
  for(k=0; k<n; ++k)
    MOSEKCALL(res,MSK_putaij(task,1+k,j,G[k][j]));

  MOSEKCALL(res,MSK_putvarbound(task,j,MSK_BK_LO,0.0,MSK_INFINITY));
}
```

```
/* t variables. */
for(j=0; j<n; ++j)
{
  MOSEKCALL(res,MSK_putaij(task,1+j,n+j,-1.0));
  MOSEKCALL(res,MSK_putvarbound(task,n+j,MSK_BK_FR,-MSK_INFINITY,MSK_INFINITY));
}

/* s variable. */
MOSEKCALL(res,MSK_putvarbound(task,2*n,MSK_BK_FR,-MSK_INFINITY,MSK_INFINITY));

sub[0] = 2*n;
for(j=0; j<n; ++j)
  sub[j+1] = n+j;
MOSEKCALL(res,MSK_appendcone(task,MSK_CT_QUAD,0.0,n+1,sub));

MOSEKCALL(res,MSK_putobjsense(task,MSK_OBJECTIVE_SENSE_MAXIMIZE));
MOSEKCALL(res,MSK_putintparam(task,MSK_IPAR_LOG,0));
```

```c
  for(k=0; k<numalpha; ++k)
  {
    MOSEKCALL(res,MSK_putcj(task,2*n,-alphas[k]));

    /* MOSEKCALL(res,MSK_writedata(task,"dump.opf")); */

    MOSEKCALL(res,MSK_optimize(task));

    /* MSK_solutionsummary(task,MSK_STREAM_MSG); */

    if ( res==MSK_RES_OK )
    {
      double er=0.0,xj;

      for(j=0; j<n; ++j)
      {
              MOSEKCALL(res,MSK_getxxslice(task,MSK_SOL_ITR,j,j+1,&xj));
              er += r[j]*xj;
      }

      MOSEKCALL(res,MSK_getxxslice(task,MSK_SOL_ITR,2*n,2*n+1,&xj));

      printf("%e %e %e\n",alphas[k],er,xj);
    }
  }

  free(sub);

  return ( 0 );
}
```

Building instructions

Using Visual studio command line tools:

```
cl portfolio.c
   /I "\program files\mosek\7\tools\platform\win64x86\h"
   /link "\program files\mosek\7\tools\platform\win64x86\bin\mosek64_7_0.lib"
```

on one line.

Running:

```
portfolio
0.000000e+000 1.073000e-001 7.217338e-001
1.000000e-001 1.073000e-001 1.667000e-001
2.000000e-001 1.073000e-001 1.667000e-001
3.000000e-001 8.053969e-002 6.822048e-002
4.000000e-001 7.195059e-002 4.232918e-002
5.000000e-001 6.975570e-002 3.735526e-002
7.500000e-001 6.766020e-002 3.382745e-002
1.000000e+000 6.679036e-002 3.281117e-002
1.500000e+000 6.598434e-002 3.213941e-002
2.000000e+000 6.560097e-002 3.191621e-002
3.000000e+000 6.522112e-002 3.175819e-002
1.000000e+001 6.469785e-002 3.164510e-002
```

Debugging tips

Use writedata:

```
MSK_writedata(task,"dump.opf");
```

Content of dump.opf:

```
[objective maximize]
   1.073e-001 x0000 + 7.37e-002 x0001 + 6.270000000000001e-002 x0002 - 1e+001 x0006
[/objective]

[constraints]
  [con c0000]  x0000 + x0001 + x0002 = 1e+000 [/con]
  [con c0001]  1.667e-001 x0000 + 2.32e-002 x0001 + 1.3e-003 x0002 - x0003 = 0e+000 [/con]
  [con c0002]  1.033e-001 x0001 - 2.2e-003 x0002 - x0004 = 0e+000 [/con]
  [con c0003]  3.38e-002 x0002 - x0005 = 0e+000 [/con]
[/constraints]

[bounds]
  [b]          0 <= * [/b]
  [b]               x0003,x0004,x0005,x0006 free [/b]
  [cone quad k0000] x0006, x0003, x0004, x0005 [/cone]
[/bounds]
```

Debugging tips

Use writedata:

```
MSK_writedata(task,"dump.opf");
```

Content of dump.opf:

```
[objective maximize]
   1.073e-001 x0000 + 7.37e-002 x0001 + 6.2700000000000001e-002 x0002 - 1e+001 x0006
[/objective]

[constraints]
  [con c0000]  x0000 + x0001 + x0002 = 1e+000 [/con]
  [con c0001]  1.667e-001 x0000 + 2.32e-002 x0001 + 1.3e-003 x0002 - x0003 = 0e+000 [/con]
  [con c0002]  1.033e-001 x0001 - 2.2e-003 x0002 - x0004 = 0e+000 [/con]
  [con c0003]  3.38e-002 x0002 - x0005 = 0e+000 [/con]
[/constraints]

[bounds]
  [b]          0 <= * [/b]
  [b]               x0003,x0004,x0005,x0006 free [/b]
  [cone quad k0000] x0006, x0003, x0004, x0005 [/cone]
[/bounds]
```

Tuning tips

Reduce looping:

Use

```
MSK_putcslice(task,0,n,r);
```

instead of

```
for(j=0; j<n; ++j)
  MSK_putcj(task,j,r[j]);
```

Optimizer API summary

- Harder to code against the optimizer API than the Fusion API.
- Highly efficient. Particularly for change and reoptimize.
- Optimizer API available for Java, .NET and Python too.

Summary

MOSEK features:

- Powerful optimization engine.
- Many interfaces included.
- Extensive documentation available.
- Fusion API is easier to use than optimizer API.

Slides!

- http://mosek.com/resources/presentations/

Summary

MOSEK features:

- Powerful optimization engine.
- Many interfaces included.
- Extensive documentation available.
- Fusion API is easier to use than optimizer API.

Slides!

- `http://mosek.com/resources/presentations/`